Cambridge Assessment International Education

Cambridge International AS & A Level

COMPUTER SCIENCE **9608/42**

Paper 4 Written Paper **October/November 2020**

MARK SCHEME

Maximum Mark: 75

Published

This mark scheme is published as an aid to teachers and candidates, to indicate the requirements of the examination. It shows the basis on which Examiners were instructed to award marks. It does not indicate the details of the discussions that took place at an Examiners' meeting before marking began, which would have considered the acceptability of alternative answers.

Mark schemes should be read in conjunction with the question paper and the Principal Examiner Report for Teachers.

Cambridge International will not enter into discussions about these mark schemes.

Cambridge International is publishing the mark schemes for the October/November 2020 series for most Cambridge IGCSE™, Cambridge International A and AS Level and Cambridge Pre-U components, and some Cambridge O Level components.

[Turn over

Generic Marking Principles

These general marking principles must be applied by all examiners when marking candidate answers. They should be applied alongside the specific content of the mark scheme or generic level descriptors for a question. Each question paper and mark scheme will also comply with these marking principles.

GENERIC MARKING PRINCIPLE 1:

Marks must be awarded in line with:

- the specific content of the mark scheme or the generic level descriptors for the question
- the specific skills defined in the mark scheme or in the generic level descriptors for the question
- the standard of response required by a candidate as exemplified by the standardisation scripts.

GENERIC MARKING PRINCIPLE 2:

Marks awarded are always **whole marks** (not half marks, or other fractions).

GENERIC MARKING PRINCIPLE 3:

Marks must be awarded **positively**:

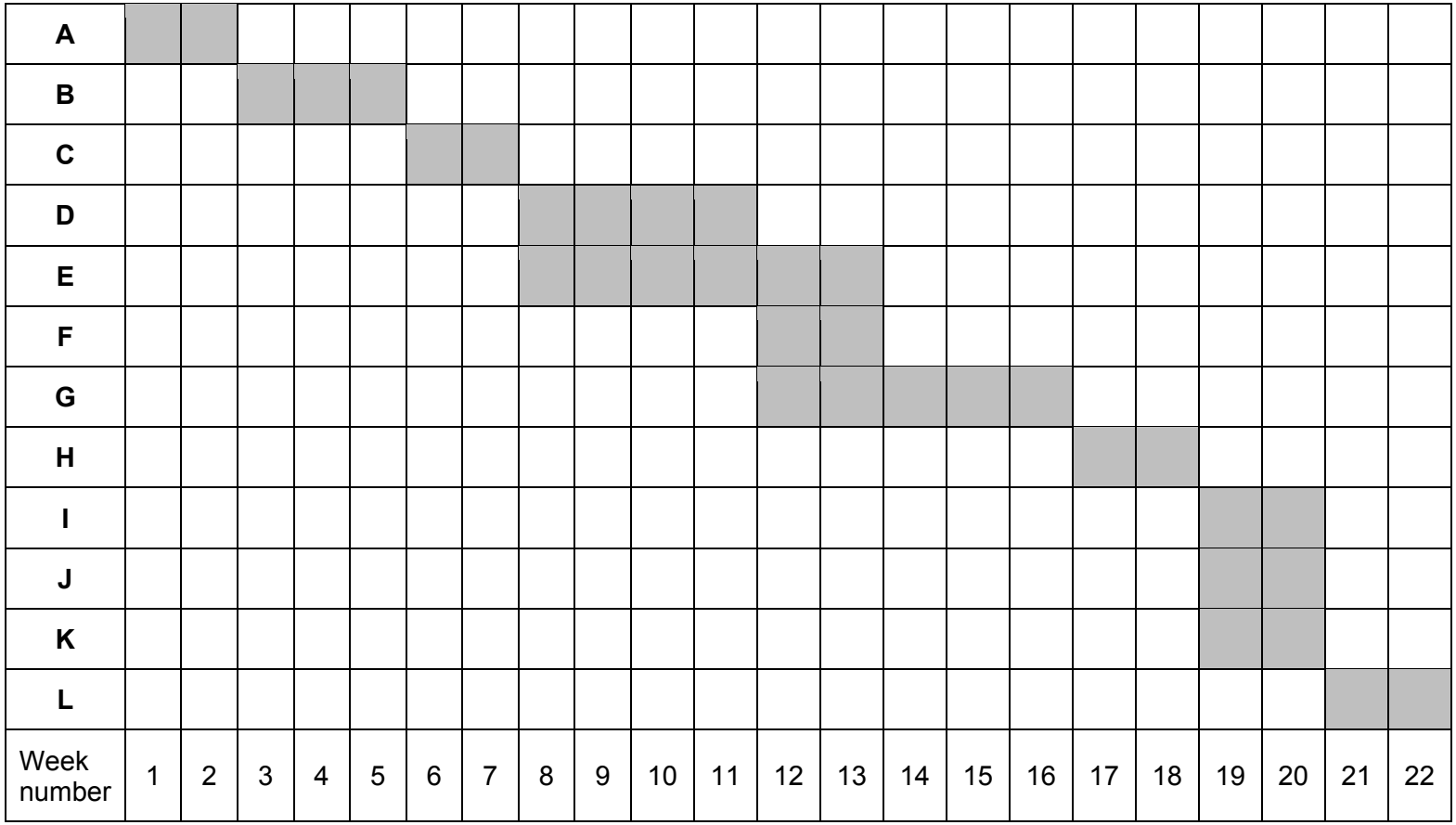
- marks are awarded for correct/valid answers, as defined in the mark scheme. However, credit is given for valid answers which go beyond the scope of the syllabus and mark scheme, referring to your Team Leader as appropriate
- marks are awarded when candidates clearly demonstrate what they know and can do
- marks are not deducted for errors
- marks are not deducted for omissions
- answers should only be judged on the quality of spelling, punctuation and grammar when these features are specifically assessed by the question as indicated by the mark scheme. The meaning, however, should be unambiguous.

GENERIC MARKING PRINCIPLE 4:

Rules must be applied consistently, e.g. in situations where candidates have not followed instructions or in the application of generic level descriptors.

GENERIC MARKING PRINCIPLE 5: Marks should be awarded using the full range of marks defined in the mark scheme for the question (however; the use of the full mark range may be limited according to the quality of the candidate responses seen).
GENERIC MARKING PRINCIPLE 6: Marks awarded are based solely on the requirements as defined in the mark scheme. Marks should not be awarded with grade thresholds or grade descriptors in mind.

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>1(a)</td><td>• Bubble (sort)
• Insertion (sort)</td><td>2</td></tr>
<tr><td>1(b)(i)</td><td>
<table>
<tr><th>LowerBound</th><th>UpperBound</th><th>ValueFound</th><th>ValueToFind</th><th>MidPoint</th></tr>
<tr><td>0</td><td>9</td><td>FALSE</td><td>21</td><td>4</td></tr>
<tr><td></td><td>3</td><td></td><td></td><td>1</td></tr>
<tr><td>2</td><td></td><td></td><td></td><td>2</td></tr>
<tr><td></td><td></td><td>TRUE</td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
</table>
One mark for columns 1 and 2, 1 mark for columns 3 and 4, 1 mark for column 5</td><td>3</td></tr>
<tr><td>1(b)(ii)</td><td>Binary (search)</td><td>1</td></tr>
<tr><td>1(b)(iii)</td><td>• 3</td><td>1</td></tr>
<tr><td>1(b)(iv)</td><td>• 1 // 2</td><td>1</td></tr>
<tr><td>1(b)(v)</td><td>• If <code>UpperBound</code> and <code>LowerBound</code> are the same // if value is on the upper bound or lower bound // if there is only 1 item in the list …
• … the last value is not checked // it won't be found // the while loop doesn't checks the last value</td><td>2</td></tr>
<tr><td>1(b)(vi)</td><td>• List is not sorted // Binary search only works on a sorted list
• 2 is less than the midpoint // 2 is after a larger value // by example
• … so (13 to) 2 would be discarded after first comparison // it will be looking for 2 in the lower half // value looking for will be discarded in first comparison</td><td>2</td></tr>
</table>

Question	Answer	Marks
2(a)	 • A(2), B(3) following A, C(2) following B • D(4) following C, E(6) following C • F(2) following D, G(5) following D • H(2) following G, L(2) following K • I, J, K (2 each) following I	5
2(b)(i)	A, B, C, D, G, H, K, L	1

Question	Answer	Marks
2(b)(ii)	I, J, K // White-box, black-box, user testing // E, F, G // Graphics development, Focus group, Program remaining levels	1
2(c)	PERT	1

Question	Answer	Marks
3(a)	• `person(clive).` • `animal(guinea_pig).` • `has_pet(clive, guinea_pig).` • `has_pet(clive, gecko).`	4
3(b)	`gecko, cat`	1

3(c)	• `wants_pet(Z, Y)`
	• `person(Z) // animal(Y)`
	• `AND animal(Y) // AND person(Z)`
	• `AND NOT`
	• `has_pet(Z, Y)`
		5

Question	Answer	Marks

4(a)	• Correct header and close (where applicable) with one parameter (ignore other parameters)
	• parameter (any identifier) assigned to attribute `FoodID`
	• Correct values assigned to `Name` (`""`) **and** `Calories` (`0`)
	
	PYTHON
	`def __init__(self, NewFoodID):`
	`  self.__FoodID = NewFoodID`
	`  self.__Name = ""`
	`  self.__Calories = 0`
	
	PASCAL
	`Constructor FoodItem.Create(NewFoodID : String);`
	`Begin`
	`  FoodID := NewFoodID;`
	`  Name : = "";`
	`  Calories := 0;`
	`End;`
	
	VB
	`Public Sub New(ByVal NewFoodID As String)`
	`  FoodID = NewFoodID`
	`  Name = ""`
	`  Calories = 0`
	`End Sub`
		3

Question	Answer	Marks

4(b)	• Correct function header and close (where applicable) with no parameter (if they have a return data type it must be correct (Integer), but not necessary)
	• Returns `Calories` without other input/assignment (using return command, or assigning to `GetCalories`)
	
	PYTHON
	`def GetCalories(self):`
	`  return(self.__Calories)`
	
	PASCAL
	`Function FoodItem.GetCalories() : Integer;`
	`Begin`
	`  GetCalories := Calories;`
	`End`
	
	VB
	`Public Function GetCalories() As Integer`
	`  Return Calories`
	`End function`
		2

4(c)	• Correct **function** header (and close) with **one** parameter passed (ignore additional parameters) (if they have a return data type it must be correct (Boolean), but not necessary)
	• Checks **parameter** is an integer between 0/1 **and** less than 2000.
	• …**Returns** true if parameter is valid **and** assigns **parameter** to `Calories`
	• …**Returns** false if invalid **and** does not assign the parameter to `Calories`
	
	`FUNCTION SetCalories(NumCalories : INTEGER) RETURNS BOOLEAN`
	`  DECLARE Valid : BOOLEAN`
	`  IF NumCalories > 0 AND NumCalories < 2000`
	`  THEN`
	`  Calories ← NumCalories`
	`  Valid ← TRUE`
	`  ELSE`
	`  Valid ← FALSE`
	`  ENDIF`
	`  RETURN Valid`
	`ENDFUNCTION`
		4

Question	Answer	Marks
4(d)(i)	Two from: • Limits access to given/set/get methods only // can only be accessed through the methods • …attributes cannot be **accidentally** changed // ensure attribute integrity/security against **accidental** change (not program) • Use of set method allows for validation of attribute • .. ensure attribute not set to inappropriate value // make sure attribute value is valid • Ensures encapsulation	**2**
4(d)(ii)	Two from: • Child class can use/has the attributes/methods of its parent class (Accept transferring attributes/methods. • The class `DailyCalories` inherits (attributes/methods) from the class `CustomerProfile` • `DailyCalories` can **use/extend** the attributes/methods from `CustomerProfile` // by example	**2**
4(d)(iii)	Two from: • Child class method/attribute can **override** parent class method/attribute // related / parent and child class have same method that has **different functions/purpose** • `GetTotalCalories()`/`SetTotalCalories()` method from `CustomerProfile` **overwritten**/has **different function** in `DailyCalories` • `TotalCalories` in `DailyCalories` **overrides** `TotalCalories` in `CustomerProfile`	**2**
4(e)	Two from: • Writing a program as a sequence of (explicit) steps/commands // sequence of events/steps // step-by-step instructions • … to gain a required outcome/result // focus is on how to achieve a result / solve a problem • The statements in the program manipulate the data • An example would be procedural programming	**2**
4(f)(i)	• Integration testing	**1**
4(f)(ii)	• Acceptance testing	**1**

Question	Answer	Marks
4(f)(iii)	Two from: • Test number • Type of test // type of test data • Test description • Expected outcome	**2**

Question	Answer	Marks

5

Label	Op Code	Operand	Comment	
	LDR	#0	// initialise IX to zero	[1]
	LDM	#0	// initialise LENGTH	[1]
	STO	LENGTH		
LOOP:	IN		// input character	[1]
	CMP	FULLSTOP	// is character a FULLSTOP (.) ?	[1]
	JPE	ENDP	// jump to ENDP if TRUE	
	STX	MESSAGE	// store character in MESSAGE + contents of IX	[1]
	INC	IX	// increment IX	[1]
	LDD	LENGTH	// increment LENGTH	[1]
	INC	ACC		
	STO	LENGTH		
	JMP	LOOP	// jump to LOOP	[1]
ENDP:	END		// end program	
LENGTH:				
FULLSTOP:	B01100000		// ASCII code for a full stop (.)	
MESSAGE:				

8

Question	Answer	Marks
6(a)	• A to B to D including NULL pointer in D … • … C not present // C present but nothing pointing to it	2
6(b)	• Added to free space/list // free pointer points to C // last element in free space links to C	1
6(c)	• Does not point to another node/address // end of list // end pointer	1

6(d)	• Correct **function** header (and close), (sensible) parameter (Not boolean) (and return data type)
	• Starting pointer set using `StartPointer`
	• Check if current pointer is NULL
	• Check if data at current pointer = parameter
	• Updates/follows next pointer to current item's pointer
	• Recursion or iteration used to check all values **not** linear search
	• Returns correct pointer when value found
	• Returns −1 when all items check and still not found
	
	`FUNCTION FindValue(Value : INTEGER) RETURNS INTEGER`
	`  CurrentPointer ← StartPointer`
	`  WHILE CurrentPointer <> NULL AND LinkedList[CurrentPointer].Data <> Value`
	`  CurrentPointer ← LinkedList[CurrentPointer].Pointer`
	`  ENDWHILE`
	`  IF LinkedList[CurrentPointer].Data = Value`
	`  THEN`
	`  RETURN CurrentPointer`
	`  ELSE`
	`  RETURN -1`
	`  ENDIF`
	`ENDFUNCTION`
		8

Question	Answer	Marks
6(e)	Four from a single ADT (one for identifying and three for description): e.g. • Stack – Linear structure – Last in first out structure – Has top and base stack pointers – Uses push to add items to top of stack – Uses pop to remove items from top of stack • Queue – Linear structure – First in first out structure – Has start and end of queue pointers – Can be circular – Uses enqueue to add item to end of queue – Uses dequeue to remove item from start of queue • Binary tree – Each node can have up to **two** (child) nodes – Parent node is above, and child nodes follow – Each node contains the data and pointer(s) – Has a root node – Can have leaf nodes – Can be output/searched in-order/post-order/pre-order – Can be ordered or unordered – Description of adding a new node // Description of ordered tree • Class – A class represents an object – Objects are instances of classes – (An object) has attributes and methods – Classes can be inherited • Hash table – Key calculated from value – .. (key) that represents a location // stores values in key locations – Key used to access location – Description of managing collisions	**4**